

CURSO FULL STACK DEVELOPER

SECCIÓN 1: HTML

ESTRUCTURA GENERAL DE HTML

HTML

HTML es un lenguaje de marcado de hipertexto. Un lenguaje de marcado o de marcas es una forma de codificar un documento que junto con el texto, incorpora etiquetas o marcas que contienen información adicional de la estructura del texto o su presentación. En otras palabras, HTML es un lenguaje de marcado porque se escribirá código a través de etiquetas y atributos.

ESTRUCTURA

<h1></h1> es un título. Es una etiqueta que puede tener un atributo. El atributo podría ser "class=".

<h1 class="text"></h1> es una etiqueta con un atributo.

HTML y CSS no son lenguajes de programación porque en los lenguajes de programación se puede incluir lógica. HTML y CSS son lenguajes de marcado.

<!DOCTYPE html> nos indica qué versión de html estamos utilizando.

><!--version--> esto es un comentario (el comentario sería "version").

<html lang="en"></html> indica el inicio y cierre del documento. Lo que está dentro de él es el contenido. "en" significa que el documento tendrá el idioma de inglés.

<head></head> es la cabecera del documento. Lleva el título del documento que es lo único que se visualiza en el documento html.

<meta charset="UTF-8"> indica el tipo de decodificación de caracteres.

`<meta name="viewport" content="width=device-width, initial-scale=1.0">` indica los cambios de dimensiones del documento.

`<link rel="stylesheet" href="">` Se utiliza para enlazar un documento externo al archivo html. Principalmente se utiliza para enlazar un documento de css.

`<body></body>` indica el contenido del documento que se mostrará luego en el navegador.

Títulos: del h1 al h6 los títulos van disminuyendo el tamaño.

`<i></i>` esto es para poner la frase itálica o cursiva

`` esto es para poner la frase en negrita

`` también se usa para negrita

`
` es un salto de línea

`` es una herramienta clave para marcar y personalizar partes específicas de contenido sin alterar la estructura del documento.

IMÁGENES

**** sirve para colocar imágenes. src muestra la dirección o ruta de la imagen y alt nos muestra lo que dice allí si no hay ninguna imagen.

width="" nos muestra la anchura de una imagen y **height=""** nos muestra el alto.

ENLACES

<[ir](https://www.primevideo.com)> es la forma en la cual se realiza un enlace. Cuando hacemos clic en el enlace, salimos de la página web en la que estábamos y se carga el enlace en el que hicimos "clic". Para evitar esto (es decir, para que el enlace se abra en una nueva pestaña), debemos colocar **target="_blank"**.

```
<h3>IMAGEN ENLACE</h3>
<a
  href="https://primevideo.com/region/na/detail/0H6VVUHOWS7SG1XZDNAVUX03C
  H?ref_=atv_plr_landingpage_play"
  target="_blank">
  </a>
```

Esta es una imagen enlace. Si hacemos clic sobre la imagen, nos abre una nueva pestaña (_blank) y nos lleva hacia el enlace indicado en href.

MENÚS

**** esto es un menú. Los ítems que estén por debajo del menú van a estar DESORDENADOS o, dicho de otra forma, sin orden numérico.

**** esto es un menú con ítems ordenados numéricamente.

**** estos son submenús o ítems

```
<ul>
    <li>Item #1</li>
    <li>Item #2</li>
    <li>Item #3</li>
    <li>Item #4</li>
    <li>Item #5</li>
</ul>

<ol>
    <li>Item #1</li>
    <li>Item #2</li>
    <li>Item #3</li>
    <li>Item #4</li>
    <li>Item #5</li>
</ol>
```

También podemos colocar enlaces en la lista:

```
<ol>
    <li><a href="https://www.primevideo.com/region-na/detail/0RIG4XTOZLID49T3X7UVK9TDEP/ref=atv_hm_mys_c_4I2SRv_2_2" target="_blank">Item #1</a> </li>
    <li>Item #2</li>
    <li>Item #3</li>
    <li>Item #4</li>
    <li>Item #5</li>
</ol>
```

Además, podemos hacer una sublista de un ítem de la lista principal:

```
<ol>
    <li><a href="https://www.primevideo.com/region/na/detail/0RIG4XTOZLID49T3X7UVK9TDEP/ref=atv_hm_mys_c_4I2SRv_2_2" target="_blank">Item #1</a> </li>
    <li>Item #2</li>
    <li>Item #3</li>
        <ul>
            <li>Item #3.1</li>
            <li>Item #3.2</li>
        </ul>
    <li>Item #4</li>
    <li>Item #5</li>
</ol>
```

Se vería de esta forma:

- Item #1
 - Item #2
 - Item #3
 - Item #3.1
 - Item #3.2
 - Item #4
 - Item #5
1. [Item #1](https://www.primevideo.com/region/na/detail/0RIG4XTOZLID49T3X7UVK9TDEP/ref=atv_hm_mys_c_4I2SRv_2_2)
 2. Item #2
 3. Item #3
 - Item #3.1
 - Item #3.2
 4. Item #4
 5. Item #5

Ideas para los menús:

```
#menu ul li :hover {
    background-color: lightslategrey;
    cursor: pointer;
}

#menu ul li: hover a {
    color: white;
}
```

La pseudo-clase hover en CSS se usa para aplicar estilos a un elemento cuando el usuario pasa el cursor por encima de él. No afecta el diseño hasta que se activa con el movimiento del mouse.

La propiedad cursor: pointer en CSS cambia el estilo del cursor cuando el usuario pasa el mouse sobre un elemento, mostrando la mano típica de los enlaces. Se usa para indicar que algo es interactivo.

TABLAS

Siempre para indicar que vamos a construir una tabla, debemos colocar `<table></table>`

`<tr></tr>` esto es para indicar las filas

`<td></td>` esto es para indicar las columnas

`<th> </th>`(Table Header - Encabezado de tabla): Se usa en lugar de `<td>` cuando queremos indicar que la celda es un título o encabezado. Por defecto, el texto dentro de `<th>` aparece en negrita y centrado.

Ejemplo:

```
<table border="1">
  <tr>
    <th>Producto</th>
    <th>Precio</th>
    <th>Cantidad</th>
  </tr>
  <tr>
    <td>Celular</td>
    <td>$500</td>
    <td>10</td>
  </tr>
  <tr>
    <td>Tablet</td>
    <td>$300</td>
    <td>5</td>
  </tr>
  <tr>
    <td>Laptop</td>
    <td>$800</td>
    <td>3</td>
  </tr>
</table>
```

Explicación del código:

`<table border="1">` → Crea una tabla y le agrega un borde (opcional).

`<tr>` → Define una fila en la tabla.

`<th>` → Crea los encabezados de la tabla ("Producto", "Precio" y "Cantidad").

`<td>` → Representa cada celda con los datos de los productos.

Pongamos de ejemplo las siguientes tablas:

```
<table border="1">

    <tr> <!-- Se debe encerrar los <th> dentro de <tr> -->

        <th>Nombre</th>

        <th>Apellido</th>

        <th>DNI</th>| 

    </tr>

    <tr><!-- Se agrega un <tr> para la fila de datos -->

        <td>Karen</td>

        <td>Clausen</td>

        <td>1</td>

    </tr>

</table>

<h2>Lista de Productos</h2>

<table border="1">

    <tr>

        <th>Producto</th>

        <th>Precio</th>

        <th>Cantidad</th>

    </tr>

    <tr>

        <td>Celular</td>

        <td>$500</td>

        <td>10</td>

    </tr>
```

```

</tr>

<tr>
    <td>Tablet</td>
    <td>$300</td>
    <td>5</td>
</tr>

<tr>
    <td>Laptop</td>
    <td>$800</td>
    <td>3</td>
</tr>

</table>

```

El resultado en el sitio web sería el siguiente:

Nombre	Apellido	DNI
Karen	Clausen	1

Lista de Productos

Producto	Precio	Cantidad
Celular	\$500	10
Tablet	\$300	5
Laptop	\$800	3

FORMULARIOS

Para crear un formulario, la estructura principal es la siguiente:

```
<form action="" method="GET">

    <label for="nom">Ingrese su nombre</label>

    <br>

    <input type="text" placeholder="Por ejemplo: Karen" maxlength="8"
minlength="1" name="nombres" id="nom" size="20"> <!--tener en cuenta
que los espacios cuentan como caracteres-->

    <!--con size cambiamos el tamaño de la barra del input para que sea
mas grande o mas pequeña-->

</form>
```

action="" sirve para agregar una ruta que nos lleva a otro archivo en el que vamos a agregar información relacionada al formulario.

method="GET" nos indica la manera en la que se enviarán esos datos. Existe GET y POST.

<label for=""></label> esta etiqueta nos visualiza el título del campo del formulario

value="" Define un valor predeterminado dentro del campo de entrada. Si el usuario no cambia el valor, el formulario enviará este dato.

placeholder="" Muestra un mensaje de ayuda dentro del campo, pero desaparece cuando el usuario escribe. No se envía como valor en el formulario si el usuario no escribe nada.

El **id="nom"** se puede usar para relacionar el **<label>** con el **<input>** mediante **for="nom"**. Cuando el usuario haga clic en el texto "Ingrese su nombre", el cursor se posicionará automáticamente en el campo de entrada.

Ejemplos:

```
<label for="nom">Ingrese su nombre</label>

    <br>
```

```
<input type="text" placeholder="Por ejemplo: Karen" maxlength="8"  
minlength="1" name="nombres" id="nom" size="20"> <!--tener en cuenta  
que los espacios cuentan como caracteres-->  
  
<!--con size cambiamos el tamaño de la barra del input para que sea  
mas grande o mas pequeña-->  
  
<br><br>  
  
<label>Ingrese su email:</label>  
  
<br>  
  
<input type="email" name="email" id="email">  
  
<br><br>  
  
<label>Ingrese su password:</label>  
  
<br>  
  
<input type="password" name="password">  
  
<br><br>  
  
<label>Ingrese su edad:</label>  
  
<br>  
  
<input type="number" name="edad" id="edad" max="100" min="18">  
  
<br><br>
```

La salida por consola sería la siguiente:

Ingrese su nombre

Por ejemplo: Karen

Ingrese su email:

Ingrese su password:

Ingrese su edad:

Radio y Checkbox

Cuando la selección es excluyente, utilizamos “radio”.

Cuando la selección puede involucrar a más de una opción, utilizamos “checkbox”.

Ejemplo de radio:

```
<label>Sexo:</label>
<br>
<input type="radio" name="sexo" value="masculino"> Masculino
<input type="radio" name="sexo" value="femenino"> Femenino
```

La salida por consola sería la siguiente:

Sexo:
○ Masculino ○ Femenino

Button y Submit

Podemos agregar un button para enviar los datos luego de seleccionar todo lo visto anteriormente. Quedaría de la siguiente forma:

```
<br>
<input type="button" value="Enviar">
```

Sexo:
○ Masculino ○ Femenino

Para que los datos se envíen, es necesario agregar un input de tipo “submit”:

```
<input type="submit" value="submit">
```

Los datos se enviarán a la dirección especificada al inicio del formulario en el `action=""`.

Select y Combobox

El elemento `<select>` se usa para crear un menú desplegable en un formulario, permitiendo que el usuario elija una opción de una lista predefinida.

1. `<select>` (*Menú desplegable estándar en HTML*)

- Muestra una lista de opciones predefinidas.
- El usuario solo puede elegir entre las opciones disponibles.
- No permite escribir una opción personalizada.
- Se usa para formularios donde hay opciones fijas.

2. `combobox` (*Menú desplegable con opción de escritura*)

- Es un tipo de select que permite al usuario escribir su propia opción o elegir una existente.
- En HTML estándar no existe un elemento `<combobox>`.
- Para crearlo en HTML5, se usa `<input type="text">` con `<datalist>`.

Ejemplo:

```
<label>Ingrese su país de residencia:</label>

<br>

<select name="pais" id="">

<option value="Argentina">Argentina</option>

<option value="Brasil">Brasil</option>

<option value="Chile">Chile</option>

<option value="Colombia">Colombia</option>

<option value="Uruguay">Uruguay</option>

<option value="Paraguay">Paraguay</option>

<option value="Venezuela">Venezuela</option>
```

```
</select>
```

Se visualiza por pantalla de la siguiente forma:

Ingrese su país de residencia:



Color

Crea una paleta de colores:

```
<input type="color" name="color" id="">
```

Range

Se utiliza para elegir el rango.

```
<input type="range" name="rango" id="">
```

ESTRUCTURA COMPLETA DE

HTML

HEADER

El elemento <header> en HTML se usa para definir la cabecera de una página web o de una sección dentro de ella.

Características de <header>:

- Se usa generalmente al inicio de una página o una sección.
- Puede contener logotipos, menús de navegación, títulos o imágenes destacadas.
- Se puede usar varias veces en una misma página, por ejemplo, en cada sección importante.
- No está limitado a la cabecera principal del sitio web.

Ejemplo:

```
html
Copy Edit

<header>
  <h1>Mi Página Web</h1>
  <nav>
    <ul>
      <li><a href="index.html">Inicio</a></li>
      <li><a href="nosotros.html">Nosotros</a></li>
      <li><a href="contacto.html">Contacto</a></li>
    </ul>
  </nav>
</header>
```

FOOTER

El footer es un pie de página. Por ejemplo:

```
<footer>
  <p>&copy; 2025 Mi Sitio Web. Todos los derechos reservados.</p>
</footer>
```

div y nav

1. <div> (Contenedor Genérico)

Es una etiqueta genérica que no tiene un significado semántico.

Se usa para agrupar y organizar elementos en una página.

Cuándo usarlo:

- Cuando necesitas un contenedor sin un propósito semántico específico.
- Para aplicar estilos o estructuras con CSS.

1. <nav> (Sección de Navegación)

Es una etiqueta semántica usada para definir menús de navegación.

Se usa para enlaces internos o externos dentro de una web.

Cuándo usarlo:

- Para menús o enlaces dentro de la página.
- Para mejorar la accesibilidad y SEO.

Característica	<div>	<nav>
Propósito	Genérico (sin significado)	Menús y navegación
Semántico	No	Sí
Usado para	Agrupar contenido	Menús y enlaces
Impacto en SEO	Bajo	Alto (Google lo reconoce como navegación)

ARTICLE

La etiqueta <article> se usa para contenido independiente y reutilizable, como:

- Publicaciones de un blog
- Noticias o artículos de prensa
- Comentarios o reseñas

- Productos en una tienda

ASIDE

La etiqueta <aside> se usa para contenido relacionado pero no central dentro de una página web.

Usos típicos de <aside>:

- Barras laterales con enlaces o publicidad
- Citas o información adicional
- Artículos relacionados en un blog
- Widgets o secciones secundarias

Ejemplo:

```
<article>
    <h2>Cómo hacer café</h2>
    <p>El café es una bebida popular...</p>
</article>

<aside>
    <h3>Artículos relacionados</h3>
    <ul>
        <li><a href="#">Los mejores granos de café</a></li>
        <li><a href="#">Diferencias entre café expreso y americano</a></li>
    </ul>
</aside>
```

SECCIÓN 2: CSS

CSS (Cascading Style Sheets u Hojas de Estilo en Cascada) es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en lenguaje de marcado.

En otras palabras, con CSS le damos estilo a un documento HTML o XHTML. El lenguaje puede ser aplicado a cualquier documento XML, XHTML, SVG, XUL, RSS, etc.

SELECTORES

1. Selector universal

Se aplica a todos los elementos de la página.

Usos comunes: Resetear estilos o aplicar reglas generales.

Ejemplo:

```
* {  
    font-family: Arial, Helvetica, sans-serif;  
}
```

font-family se refiere a la fuente para todo el documento. Si le colocamos más de una, entonces va a tomar la primera; si no, tomará la segunda; y así sucesivamente.

Define una lista de fuentes o familia de fuentes con un orden de prioridad para utilizar en un elemento seleccionado. A diferencia de la mayoría de las propiedades css, los valores se separan con comas (",") para indicar que son valores alternativos.

2. Selector de Etiqueta (o Tipo)

Aplica estilos a todas las instancias de una etiqueta específica.

Usos comunes: Dar estilos globales a encabezados (h1, h2), párrafos (p), listas (ul, li), etc.

Ejemplo:

```
h1 {  
    color: antiquewhite;  
    background: dimgrey;  
}
```

color: elegimos el color de la letra.

background: red es para elegir el color de fondo de la etiqueta a la cual le añadimos la propiedad.

Se vería de la siguiente forma:

Curriculum Vitae



padding: es un borde interno. Si por ejemplo establecemos un “padding: 10px”, se agranda el borde interno de la etiqueta. Ejemplo:

Curriculum Vitae



font-size es para cambiar el tamaño de la letra.

text-decoration: La propiedad text-decoration en CSS se usa para agregar o quitar efectos visuales en el texto, como subrayado, tachado, etc.

Principales valores de text-decoration:

```
p {  
    text-decoration: underline; /* Subraya el texto */  
    text-decoration: overline; /* Línea arriba del texto */  
    text-decoration: line-through; /* Texto tachado */  
    text-decoration: none; /* Quita cualquier decoración */  
}
```

3. Selector de ID (#id)

Se usa para estilizar un único elemento con un atributo id específico.

Reglas clave del ID:

- Debe ser único en la página (no se repite en otros elementos).
- Se usa más en JavaScript para manipular elementos.
- Para estilos, es mejor usar clases (.clase) en lugar de IDs.

Ejemplo:

```
#resumen {  
    border: 5px solid rgb(99, 64, 11); /*borde, tamaño, tipo, color*/  
}
```

🔥 Resumen: Cuándo usar cada uno

Selector	Se aplica a...	Símbolo	Uso común
Universal	Todos los elementos	*	Resetear estilos generales
Etiqueta	Todos los de un tipo	h1 , p , div	Dar estilos a etiquetas específicas
ID	Un solo elemento	#id	Casos específicos o manipulación con JS

4. Selector de clase

El selector de clase se usa para aplicar estilos a múltiples elementos que comparten una misma clase. Se define con un punto (.) seguido del nombre de la clase.

Ejemplo:

HTML

```
<article id="cursos">

    <h2>Cursos</h2>

    <h3><i>Full Stack Web Development | Udemy</i></h3>

        <p class="parrafo1"><b>Frontend:</b> HTML5, CSS3 (Flexbox, Grid), JavaScript, jQuery, Angular. Desarrollo de interfaces de usuario interactivas y dinámicas, con enfoque en experiencia de usuario (UX).</p>

        <p class="parrafo2"><b>Backend:</b> PHP, Laravel. Creación de aplicaciones escalables y seguras, con integración de servicios web y lógica de negocio.</p>

        <p class="parrafo3"><b>Bases de datos:</b> MySQL. Diseño, optimización y gestión de bases de datos relacionales, incluyendo consultas avanzadas y normalización de datos.</p>

    </article>
```

CSS

```
.parrafo2 {

}

color: rgb(174, 64, 0);
```

🔥 Diferencia con `id (#)`

Selector	Símbolo	Se aplica a...	¿Repetible?
Clase	<code>.mi-clase</code>	Varios elementos	<input checked="" type="checkbox"/> Sí
ID	<code>#mi-id</code>	Un solo elemento	<input checked="" type="checkbox"/> No (debe ser único)

5. Selector de atributo

El selector de atributo permite aplicar estilos a elementos basados en sus atributos y valores. Se usa con corchetes [].

◆ Ejemplo básico: Seleccionar elementos con un atributo específico

CSS

```
[input] {  
    border: 2px solid blue;  
}
```

 Copy  Edit

◆ Esto aplica un borde azul a todos los elementos que tengan el atributo `input`, como:

html

```
<input type="text">  
<input type="password">
```

 Copy  Edit

◆ Ejemplo: Seleccionar por valor de atributo

CSS

```
input[type="text"] {  
    background-color: lightgray;  
}
```

 Copy  Edit

◆ Esto solo afecta a los `<input>` con `type="text"`:

html

```
<input type="text" placeholder="Escribí algo...">  
<input type="password" placeholder="Contraseña">
```

 Copy  Edit

⚠ Solo el primer `<input>` tendrá fondo gris.



◆ Variantes avanzadas

Selector	Descripción	Ejemplo
[atributo]	Selecciona elementos que tienen ese atributo	<code>img[alt] {border: 1px solid red;}</code>
[atributo="valor"]	Solo los que tienen ese valor exacto	<code>a[target="_blank"] {color: red;}</code>
[atributo~= "valor"]	Contiene esa palabra exacta en una lista separada por espacios	<code>[class~="destacado"] {font-weight: bold;}</code>
[atributo^="valor"]	Empieza con ese valor	<code>a[href^="https://"] {color: green;}</code>
[atributo\$="valor"]	Termina con ese valor	<code>img[src\$=".jpg"] {border-radius: 10px;}</code>
[atributo*= "valor"]	Contiene ese valor en cualquier parte	<code>input[placeholder*="nombre"] {background: yellow;}</code>

6. Selector de Atributo Combinado con Clase o ID

Podés combinar selectores de atributos con clases (.) o IDs (#) para aplicar estilos más específicos.

◆ Selector de Atributo dentro de una Clase (.clase[atributo])

Si querés seleccionar elementos con una clase específica que también tengan un atributo, usás:

css

 Copy  Edit

```
.mi-clase[disabled] {  
    background-color: gray;  
}
```

◆ Ejemplo en HTML:

html

 Copy  Edit

```
<button class="mi-clase" disabled>Botón deshabilitado</button>  
<button class="mi-clase">Botón activo</button>
```

⚠ Solo el primer botón tendrá fondo gris porque tiene `disabled`.

◆ Selector de Atributo dentro de un ID (#id[atributo])

Si querés seleccionar un ID específico con un atributo, usás:

css

 Copy  Edit

```
#formulario[action^="https"] {  
    border: 2px solid blue;  
}
```

◆ Ejemplo en HTML:

html

 Copy  Edit

```
<Form id="formulario" action="https://example.com"></Form>  
<form id="formulario" action="http://example.com"></form>
```

👉 Solo el primer `<form>` tendrá borde azul porque `action` empieza con "https".

🚀 Resumen de Combinaciones Útiles

Selector	Descripción	Ejemplo CSS	Afecta a...
<code>.clase[atributo]</code>	Elementos con cierta clase y atributo	<code>.boton[disabled]</code>	<code><button class="boton" disabled></code>
<code>#id[atributo]</code>	Elemento con cierto ID y atributo	<code>#login-form[action^="https"]</code>	<code><form id="login-form" action="https://..."></code>
<code>.clase[atributo="valor"]</code>	Clase con atributo exacto	<code>.tarjeta[data-tipo="premium"]</code>	<code><div class="tarjeta" data-tipo="premium"></code>

7. Selector Hijo (>)

El selector hijo (>) permite seleccionar sólo los elementos que son hijos directos de otro elemento.

◆ Sintaxis del Selector Hijo

CSS

```
padre > hijo {  
    propiedad: valor;  
}
```

Ejemplo:

CSS

 Copy  Edit

```
div > p {  
    color: blue;  
}
```

◆ HTML:

html

 Copy  Edit

```
<div>|  
  <p>Este párrafo será azul ✓</p>  
  <section>  
    <p>Este NO será azul ❌ (no es hijo directo de `<div>`, sino de `<section>`)</p>  
  </section>  
</div>
```

Solo el primer `<p>` será azul porque es un hijo directo de `<div>`.

Diferencia con el Selector Descendiente (espacio):

CSS

```
div p { /* Selector descendiente */  
    color: red;  
}
```

Esto selecciona TODOS los `<p>` dentro de `<div>`, sin importar la profundidad:

```
html Copy Edit
<div>
  <p>Será rojo ✓</p>
  <section>
    <p>También será rojo ✓ (es descendiente, aunque no hijo directo)</p>
  </section>
</div>
```

Resumen:

Selector	Descripción	Afecta a...
<code>padre > hijo</code>	Solo los hijos directos	<code><div> <p>...</p> </div></code> ✓
<code>padre hijo</code> (sin <code>></code>)	Todos los descendientes	<code><div> <section> <p>...</p> </section> </div></code> ✓

Orden de Prioridad en CSS

De mayor a menor prioridad:

1. !important (Fuerza una regla por encima de cualquier otra).
2. Estilos en línea (`style="color: red;"` en HTML).
3. Selectores ID (`#id {}`): Son más específicos que las clases o etiquetas.
4. Selectores de clase, atributos y pseudoclases (`.clase {}`, `[type="text"] {}`, `:hover {}`): Tienen menor prioridad que los IDs pero mayor que los selectores de etiqueta.
5. Selectores de etiqueta y pseudoelementos (`h1`, `p`, `div`, `::before`): Son los menos específicos y tienen menor prioridad que los anteriores.
6. Selectores universales (*) y estilos heredados: Tienen la menor prioridad y se aplican solo si no hay reglas más específicas.

Reglas más específicas siempre ganan sobre reglas más generales.

Resumen final:

- !important es lo más fuerte, pero usalo con cuidado.
- Los estilos en línea (style="") tienen alta prioridad.
- IDs (#id) tienen más peso que clases (.clase).
- Si dos reglas tienen la misma especificidad, gana la última en el código.
- La cascada respeta el orden de las reglas si tienen la misma prioridad.

FUENTES EN CSS

<https://fonts.googleapis.com/>

Para importar fuentes en CSS, hay varias formas:

1. Usar @import en el CSS

Explicación:

Se coloca al inicio del archivo CSS y carga la fuente desde un enlace externo, como Google Fonts.

Ejemplo:

```
@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;700&display=swap');

body {
    font-family: 'Roboto', sans-serif;
}
```

Ventajas:

- Mantiene el HTML limpio.
- Fácil de actualizar y administrar en el CSS.

Desventajas:

- Puede afectar la velocidad de carga porque se carga después del CSS principal.

2. Usar <link> en el HTML

Se coloca en <head> para cargar la fuente antes de que se aplique el CSS.

Ejemplo:

```
html
<head>
    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;700&display=swap">
</head>
```

Y en el CSS:

```
CSS Copy Edit
body {
    font-family: 'Roboto', sans-serif;
}
```

Ventajas:

- Mejora el rendimiento porque el navegador carga la fuente antes del CSS.
- Compatible con todos los navegadores.

Desventajas:

- Requiere modificar el HTML.

3. Usar @font-face para fuentes locales

Explicación:

Se usa cuando la fuente está guardada en el servidor o en tu proyecto.

Ejemplos:

```
@font-face {
    font-family: 'MiFuente';
    src: url('fonts/MiFuente.woff2') format('woff2'),
         url('fonts/MiFuente.woff') format('woff');
    font-weight: normal;
    font-style: normal;
}

body {
    font-family: 'MiFuente', sans-serif;
}
```

```

@font-face {
    font-family: EBGaramond;
    src: url(../font/EBGaramond/EB_Garamond/EBGaramond-VariableFont_wght.ttf);
}

@font-face {
    font-family: Merriweather-Regular;
    src: url(../font/Merriweather-Regular/Merriweather-Regular.ttf);
}

h1 {
    font-family: Merriweather-Regular, serif;
    font-weight: 700;
}

h3 {
    font-family: EBGaramound, Cambria, Cochin, Georgia, Times, 'Times New Roman', serif;
}

```

Ventajas:

- No depende de servicios externos como Google Fonts.
- Más control sobre la fuente y su disponibilidad.

Desventajas:

- Aumenta el tamaño del proyecto porque hay que incluir los archivos .woff2, .woff, .ttf, etc.
- Si la fuente no está bien optimizada, puede ralentizar la carga.

4. Usar fuentes del sistema (system-ui)

Usa fuentes predeterminadas del sistema operativo del usuario.

Ejemplo:

```
body {  
    font-family: system-ui, sans-serif;  
}
```

Ventajas:

- Carga instantáneamente sin necesidad de descargar fuentes.
- Optimizado para cada sistema operativo.

Desventajas:

- No permite personalización avanzada de tipografías.

Clasificaciones de font en CSS y su uso

CSS proporciona varias propiedades para manipular la apariencia del texto. A continuación, una lista de las más importantes:

1. *font-family* → Define la fuente del texto

Permite especificar el tipo de fuente que se usará. Se pueden listar varias fuentes como respaldo en caso de que la primera no esté disponible.

2. *font-size* → Controla el tamaño del texto

Define qué tan grande será la fuente. Se puede especificar en diferentes unidades como px, em, rem, %, vw, etc.

3. *font-weight* → Define el grosor de la fuente

Permite hacer el texto más grueso o más delgado. Se puede usar valores numéricos (100 a 900) o palabras clave como normal, bold, lighter, etc.

4. font-style → Define el estilo de la fuente

Se usa para poner el texto en itálica o en estilo oblicuo.

5. font-variant → Controla variantes tipográficas

Se usa para cambiar la presentación del texto, por ejemplo, a versalitas (small-caps).

6. line-height → Controla el interlineado

Define el espacio entre líneas de texto.

7. letter-spacing → Ajusta el espacio entre letras

Permite aumentar o reducir la separación entre caracteres.

8. word-spacing → Ajusta el espacio entre palabras

Similar a letter-spacing, pero aplicado a palabras.

9. text-transform → Cambia la capitalización del texto

Permite poner el texto en mayúsculas, minúsculas o capitalizar la primera letra de cada palabra.

10. text-align → Alinea el texto

Se usa para alinear el texto a la izquierda, derecha, centro o justificarlo.

11. text-decoration → Agrega o quita subrayado, tachado, etc.

Permite decorar el texto con líneas o eliminar decoraciones predeterminadas.

12. text-shadow → Agrega sombras al texto

Permite aplicar sombras con diferentes desplazamientos y difuminados.

13. white-space → Controla cómo se manejan los espacios y saltos de línea

Se usa para evitar que el texto se divida en líneas, permitir saltos de línea o mantener espacios en blanco.

14. overflow-wrap / word-break → Maneja la ruptura de palabras

Se usa para evitar que palabras largas sobresalgan del contenedor.

15. font (Shorthand) → Agrupa varias propiedades en una sola

Permite definir varias propiedades tipográficas en una sola línea.

FILTROS EN CSS

La propiedad filter en CSS se usa para aplicar efectos visuales a imágenes, fondos y otros elementos HTML. Estos efectos pueden incluir desenfoque, brillo, contraste, escala de grises, entre otros.

Sintaxis general:

```
selector {  
    filter: tipo-de-filtro(valor);  
}
```

blur(px) → Desenfoque

Difumina el elemento con el valor en píxeles.

brightness(valor) → Brillo

Ajusta el brillo de un elemento. El valor 1 es el normal, >1 aumenta el brillo, <1 lo reduce.

```
img {  
    filter: brightness(1.5); /* Más brillante */  
}
```

contrast(valor) → Contraste

Ajusta el contraste del elemento. Un valor de 1 es normal, >1 aumenta el contraste, <1 lo reduce.

```
img {  
    filter: contrast(2); /* Duplica el contraste */  
}
```

grayscale(%) → Escala de grises

Convierte una imagen a blanco y negro. 0% es normal y 100% es completamente en escala de grises.

```
img {  
    filter: grayscale(100%);  
}
```

invert(%) → Invertir colores

Invierte los colores de la imagen. 100% es completamente invertido.

```
img {  
    filter: invert(100%);  
}
```

opacity(%) → Transparencia

Controla la opacidad del elemento. 0% es totalmente transparente y 100% es completamente visible.

```
img {  
    filter: opacity(50%);  
}
```

saturate(valor) → Saturación de colores

Ajusta la intensidad de los colores. 1 es el valor normal, >1 aumenta la saturación, <1 la reduce.

```
img {  
    filter: saturate(3); /* Aumenta la saturación 3 veces */
```

```
}
```

***sepia(%)* → Efecto sepia**

Aplica un tono marrón similar a las fotos antiguas. 0% es normal y 100% es completamente sepia.

```
img {  
    filter: sepia(100%);  
}
```

***hue-rotate(grados)* → Rotación de colores**

Cambia los colores de la imagen desplazando el matiz en grados (0° a 360°).

```
img {  
    filter: hue-rotate(180deg); /* Cambia los colores completamente */  
}
```

***drop-shadow()* → Sombra para imágenes y elementos no rectangulares**

Similar a box-shadow, pero aplicado a imágenes y elementos con transparencia.

```
img {  
    filter: drop-shadow(5px 5px 10px rgba(0, 0, 0, 0.5));  
}
```

***filter: none;* → Desactivar filtros**

Si un elemento tiene filtros aplicados, puedes desactivarlos con none.

UNIDADES DE MEDIDA

En CSS, las unidades de medida se utilizan para definir dimensiones como el ancho, alto, márgenes, paddings, fuentes, etc. Se dividen en unidades absolutas y relativas.

1. Unidades Absolutas

Son valores fijos que no dependen de otros elementos o del tamaño de la pantalla.

Unidad	Significado	Ejemplo
px (píxeles)	Unidad más usada, tamaño fijo en pantalla.	<code>width: 200px;</code>
cm (centímetros)	Basado en medidas reales.	<code>width: 5cm;</code>
mm (milímetros)	Subdivisión de cm .	<code>width: 10mm;</code>
in (pulgadas)	1 in = 2.54 cm = 96 px.	<code>width: 2in;</code>
pt (puntos)	Usado en impresión, 1pt = 1/72 in.	<code>font-size: 12pt;</code>
pc (picas)	1 pica = 12 pt.	<code>font-size: 2pc;</code>

2. Unidades Relativas

Se ajustan dinámicamente según el contexto, como el tamaño del viewport o el contenedor padre.

Unidad	Referencia	Ejemplo
em	Relativo al tamaño de fuente del elemento padre.	<code>font-size: 2em;</code> (2 veces el tamaño base)
rem	Relativo al tamaño de fuente del <code>html</code> (root).	<code>font-size: 1.5rem;</code> (1.5 veces el <code>html</code>)
%	Relativo al tamaño del contenedor padre.	<code>width: 80%;</code>

- Basadas en el tamaño de fuente:

Unidad	Referencia	Ejemplo
em	Relativo al tamaño de fuente del elemento padre.	<code>font-size: 2em;</code> (2 veces el tamaño base)
rem	Relativo al tamaño de fuente del <code>html</code> (root).	<code>font-size: 1.5rem;</code> (1.5 veces el <code>html</code>)
%	Relativo al tamaño del contenedor padre.	<code>width: 80%;</code>

```
html { font-size: 16px; }

p { font-size: 1.5rem; } /* 1.5 * 16px = 24px */
```

Diferencia entre em y rem:

1. em se multiplica por el tamaño del parente.
2. rem solo se multiplica por el tamaño de html.

- Basadas en el tamaño de la pantalla (Viewport)

Unidad	Referencia	Ejemplo
vw	1% del ancho de la pantalla (viewport width).	width: 50vw; (50% del ancho de la pantalla)
vh	1% del alto de la pantalla (viewport height).	height: 100vh; (ocupa toda la altura del viewport)
vmin	1% del lado más pequeño del viewport.	width: 50vmin;
vmax	1% del lado más grande del viewport.	width: 50vmax;

Se ajustan según el tamaño de la pantalla del usuario.

```
div {

    width: 80vw; /* 80% del ancho de la pantalla */

    height: 50vh; /* 50% del alto de la pantalla */

}
```

- Basadas en el contenido

Se ajustan según el tamaño del contenido o de los caracteres.

Unidad	Referencia	Ejemplo
ch	Ancho del carácter "0" del elemento.	<code>width: 10ch;</code> (equivalente a 10 caracteres "0")
ex	Altura de la "x" minúscula del elemento.	<code>line-height: 2ex;</code>

```
p {
    width: 50ch; /* Limita el ancho a 50 caracteres */
}
```

3. Unidades Flexibles (CSS Grid & Flexbox)

Unidad	Referencia	Ejemplo
fr	Fracción del espacio disponible en un contenedor Grid.	<code>grid-template-columns: 1fr 2fr;</code>

```
.container {
    display: grid;
    grid-template-columns: 1fr 2fr;
}
```

FORMAS DE AGREGAR COLORES

1. Nombre del color

Se usa el nombre en inglés de un color estándar.

Ejemplo:

```
h1 {  
    color: red;  
}
```

Fácil de recordar, pero con opciones limitadas.

2. Código hexadecimal (#rrggbb)

Se define el color con valores hexadecimales de rojo, verde y azul (RGB).

Ejemplo:

```
p {  
    color: #ff5733; /* Naranja */  
}
```

https://www.w3schools.com/cssref/css_colors.php

3. RGB (rgb(r, g, b))

Cada valor va de 0 a 255.

Ejemplo:

```
body {  
    background-color: rgb(34, 45, 200); /* Azul */  
}
```

<https://www.toptal.com/developers/css3maker/css3-rgba>

4. RGBA (rgba(r, g, b, a))

Incluye un valor de opacidad (a) de 0 (transparente) a 1 (opaco).

Ejemplo:

```
div {  
    background-color: rgba(255, 0, 0, 0.5); /* Rojo semitransparente */  
}
```

<https://www.toptal.com/developers/css3maker/css3-rgba>

5. transparent

Hace que el fondo sea completamente transparente.

Ejemplo:

```
div {  
    background-color: transparent;  
}
```

Estas son las opciones principales pero hay más tipos.

¿Cuál usar?

Hexadecimal o RGB → Para colores específicos.

- HSL → Para ajustes intuitivos de colores.
- RGBA o HSLA → Si necesitás transparencia.
- currentColor → Para mantener consistencia en el diseño.

6. HSL (Hue, Saturation, Lightness)

Hue (tono) va de 0 a 360, Saturation (saturación) y Lightness (luminosidad) van de 0% a 100%.

Ejemplo:

```
color: hsl(200, 100%, 50%); /* Azul fuerte */
```

7. HSLA (HSL con transparencia)

Igual que HSL, pero con un cuarto valor (a) para la opacidad.

Ejemplo:

```
background-color: hsla(200, 100%, 50%, 0.5); /* Azul con transparencia */
```

8. HWB (Hue, Whiteness, Blackness)

HWB (Hue, Whiteness, Blackness) es un modelo de color que se agregó en CSS4. Es similar a HSL, pero en lugar de definir saturación y luminosidad, usa:

H (Hue) → Matiz (de 0° a 360°).

W (Whiteness) → Cantidad de blanco (de 0% a 100%).

B (Blackness) → Cantidad de negro (de 0% a 100%).

Ejemplo:

```
color: hwb(200 30% 20%); /* Azul con 30% de blanco y 20% de negro */
```

FORMAS DE CAMBIAR EL TEXTO

```
text-indent: 15px; /*sangria entre texto y texto*/  
word-spacing: 10px; /*separacion entre palabras de 10px*/  
letter-spacing: 5px; /*separacion entre letras de 5px*/  
text-transform: uppercase; /*mayusculas*/  
text-decoration: line-through; /*tacha el texto*/  
text-decoration: underline; /*subraya el texto*/  
text-align: center; /*centra el texto*/  
text-shadow: 1px 5px 2px gray; /*sombreado*/  
box-shadow: 5px 5px 10px gray; /*sombreado para cajas o  
contenedores*/  
}
```

En el text-shadow

- 1px primer valor: si es positivo, el sombreado va hacia la derecha y si es negativo, hacia la izquierda.
- 5px segundo valor: si es positivo, el sombreado va hacia abajo y si es negativo hacia arriba.
- 2px tercer valor: es para el difuminado (si se ve más fuerte o más claro el sombreado)

BACKGROUND

En HTML y CSS, la propiedad background se usa para definir el fondo de un elemento. Puede aplicarse a div, body, section, header, y otros elementos.

Ejemplo:

```
body {  
    background-color: #f0f0f0; /* Color de fondo */  
    background-image: url('fondo.jpg'); /* Imagen de fondo */  
    background-size: cover; /* Ajuste de la imagen */  
    background-position: center; /*top bottom right left: primero y  
    segundo valor para el eje vertical y los otros dos valores para el eje  
    horizontal (lados). Pueden ser números o palabras* o palabras y números  
    combinados. Ejemplo right 10px*/  
    background-repeat: no-repeat; /* No repetir la imagen */  
}
```

Otro ejemplo:

HTML:

```
<!--PARA BACKGROUND-->
```

```
<div class="container">  
  
</div>
```

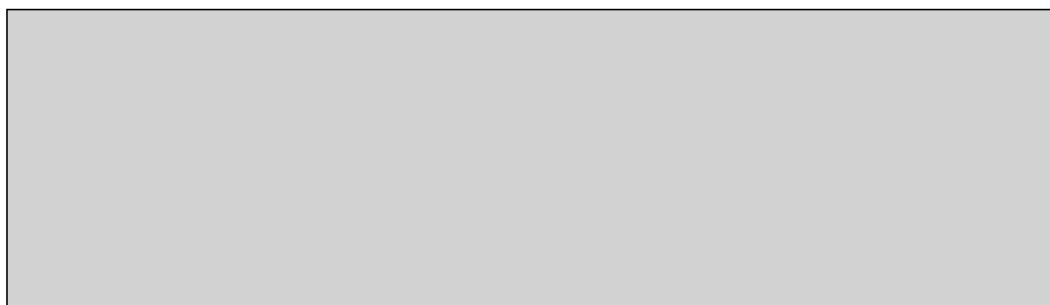
CSS:

```
/*PARA BACKGROUND*/
```

```
.container {
```

```
width: 70%; /*estilo responsive*/  
  
height: 300px;  
  
background-color: lightgray; /* Para que se vea */  
  
margin: 30px auto;  
  
border: 2px solid black; /* Para que tenga bordes */  
}
```

Se vería de la siguiente forma:



Este contenedor cubre el 70% de la pantalla, por lo que al achicar o agrandar la página, seguirá cubriendo el mismo porcentaje.

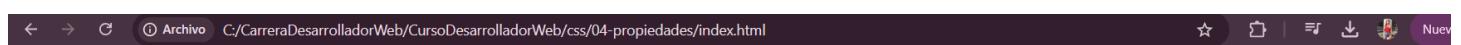


Este contenedor cubre el 70% de la pantalla, por lo que al achicar o agrandar la página, seguirá cubriendo el mismo porcentaje.

Se encuentra en el medio de la pantalla porque le colocamos un margen de 30px arriba y abajo y le especificamos que por los costados esté en “auto”. Esto ajusta automáticamente al contenedor para que quede en el medio de la pantalla. Tener en cuenta que el “auto” se aplica al 100% del contenedor, no al 70%.

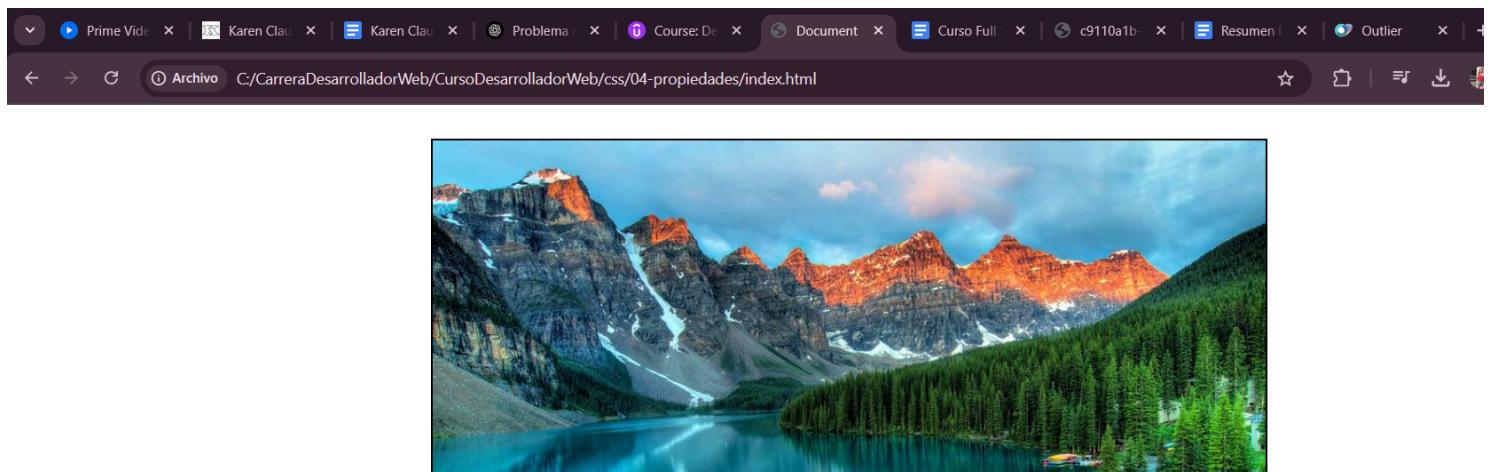
Veamos a continuación un ejemplo relacionado con el “background-size”.

Si colocamos “background-size: contain”, css intentará escalar la imagen lo más posible sin estirar:



Tener en cuenta que el background es gris.

Si le colocamos “background-size: cover”, hace que la imagen se ajuste al contenedor (por ejemplo, un div o body) de manera que siempre cubra toda el área, incluso si eso significa recortar partes de la imagen:



Diferencias entre %, vh y px

1. **vh (Viewport Height)**

- 1vh = 1% de la altura total del viewport (ventana visible del navegador).
- 100vh = ocupa el 100% del alto del viewport, sin importar el contenido dentro del elemento.
- Ventaja: Asegura que el elemento siempre tenga un tamaño relativo a la ventana, útil para fondos o secciones de pantalla completa.
- Problema en móviles: En algunos navegadores móviles, el vh puede cambiar dinámicamente si aparece o desaparece la barra de navegación del sistema.

```
.attachment {  
    width: 100%;  
    height: 100vh; /* Ocupa todo el alto de la pantalla */  
}
```

2. *px (Píxeles)*

- Define una altura fija en píxeles.
- No es responsive, lo que significa que en pantallas más pequeñas puede quedar demasiado grande o pequeño.

```
.attachment {  
    height: 500px; /* Siempre mide 500px sin importar el tamaño de la  
pantalla */  
}
```

3. *% (Porcentaje)*

- Define la altura como un porcentaje del contenedor padre.
- Importante: Si el contenedor padre no tiene una altura explícita, el % no tendrá efecto.

```
.parent {  
    height: 500px;  
}  
  
.attachment {  
    height: 50%; /* 50% del contenedor padre */  
}
```

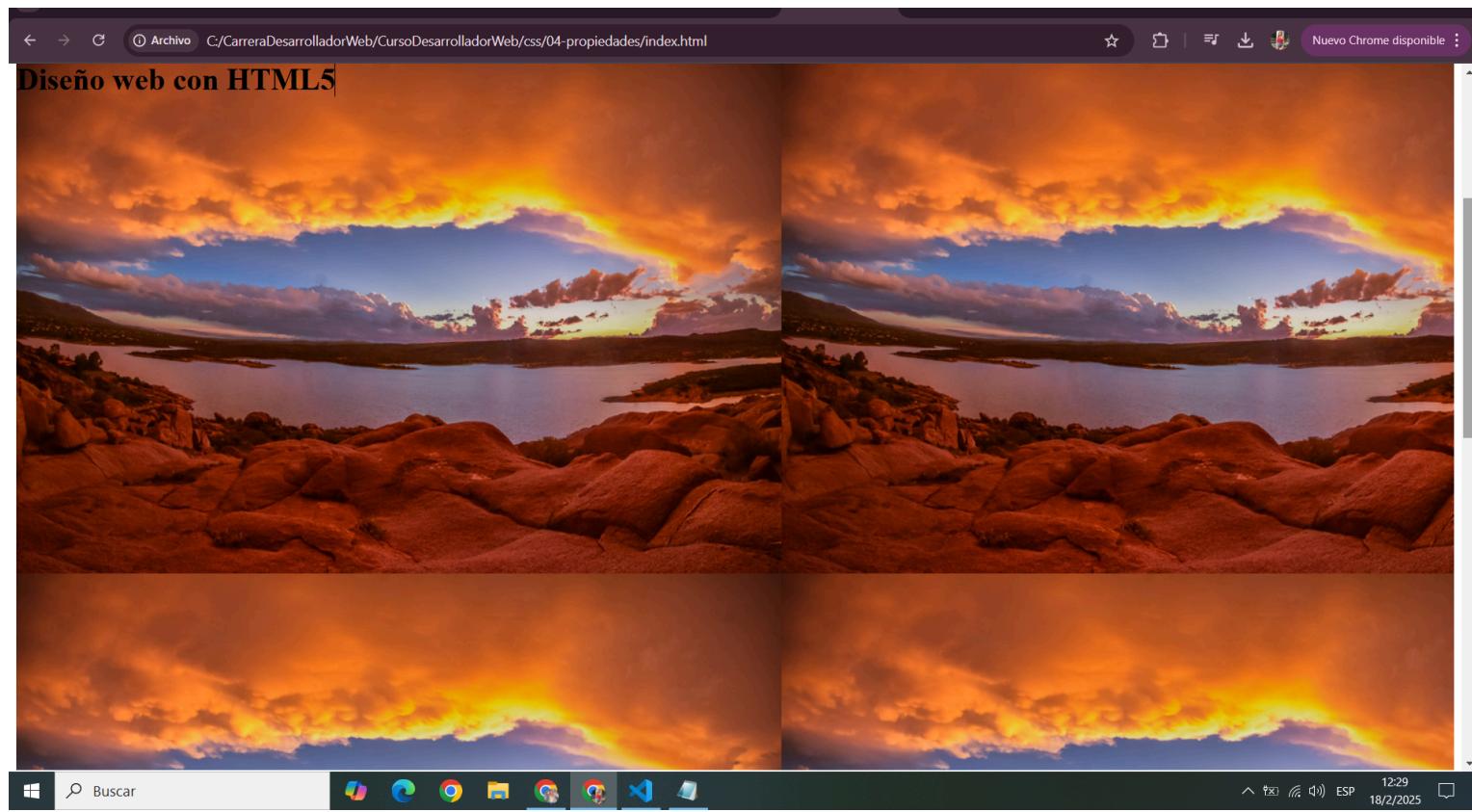
Propiedad	vh (Viewport Height)	px (Píxeles)	% (Porcentaje)
Definición	Unidad relativa al alto del viewport (ventana del navegador).	Unidad absoluta basada en píxeles físicos de la pantalla.	Unidad relativa al tamaño del contenedor padre.
Ejemplo	<code>height: 100vh;</code> (ocupa toda la altura visible del navegador)	<code>height: 500px;</code> (altura fija de 500px)	<code>height: 50%;</code> (ocupa el 50% de la altura del contenedor padre)
Responsivo	<input checked="" type="checkbox"/> Sí, cambia con el tamaño de la pantalla.	<input type="checkbox"/> No, es un valor fijo.	<input checked="" type="checkbox"/> Sí, se adapta al contenedor padre.
Ventaja principal	Garantiza que un elemento siempre tenga una altura proporcional a la pantalla.	Precisión total, útil para elementos que deben mantenerse fijos.	Se adapta dinámicamente a su contenedor.
Desventaja	Puede causar problemas en móviles si las barras de navegación cambian el tamaño del viewport.	No se ajusta a diferentes pantallas, lo que puede afectar la experiencia en dispositivos distintos.	No funciona si el contenedor padre no tiene una altura definida.
Mejor uso	Fondos de pantalla completa, secciones de altura fija relativa a la pantalla.	Elementos con altura específica como botones o íconos.	Diseños flexibles que dependen del tamaño del contenedor.

Veamos un ejemplo donde utilizamos vh y colocamos una imagen de fondo:

CSS

```
.attachment {
    width: 100%;
    height: 100vh; /* Ocupa todo el alto de la pantalla */
    background-image: url(..../img/horizontal1.jpg);
}
```

El diseño se vería de la siguiente forma:



Background attachment-fixed

La propiedad `background-attachment: fixed` hace que la imagen de fondo permanezca fija en su posición, sin moverse cuando el usuario hace scroll en la página. Es decir, el contenido se desplaza, pero el fondo se mantiene estático.

```
.clase {  
    width: 100%;  
    height: 100vh; /* Ocupa todo el alto de la pantalla */  
    background-image: url(../img/estrellas.jpg);  
    background-repeat: no-repeat;  
    background-position: center;  
    background-size: cover;
```

```
}
```



```
.dos {
```



```
    background-attachment: fixed;
```



```
}
```

La imagen de fondo no se moverá cuando el usuario haga scroll.

Se ajusta a la pantalla con background-size: cover.

Se mantiene centrada con background-position: center.

Degrado

<https://cssgradient.io/>

webgradients.com

En CSS, los degradados permiten crear transiciones suaves entre dos o más colores sin necesidad de imágenes. Se generan con background-image usando gradientes lineales, radiales o cónicos.

Tipos de degradados en CSS:

- **Degrado Lineal (*linear-gradient*)**

Es el más común y se define con una dirección y varios colores.

```
background: linear-gradient(to right, red, blue);
```

to right → El degradado va de izquierda a derecha.

red, blue → Rojo cambia gradualmente a azul.

```
background: linear-gradient(to top, crimson 30%, steelblue 70%);
```

En este caso le expresamos los porcentajes que queremos que tenga cada color.

```
background: linear-gradient(to bottom, #00008b7e, #4683b481),  
url(..../img/horizontal1.jpg);  
}
```

En este caso le agregamos una imagen de fondo al degradado.

- ***Degradado Radial (radial-gradient)***

El color cambia desde el centro hacia afuera en círculos concéntricos.

```
background: radial-gradient(circle, red, yellow, blue);
```

circle → Forma circular (también puede ser ellipse).

red → yellow → blue → Cambia desde el centro hacia afuera.

- ***Degradado Cónico (conic-gradient)***

Crea un degradado giratorio desde el centro.

```
background: conic-gradient(red, yellow, blue);
```

Los colores giran alrededor del centro, creando un efecto de "rueda".

BOX MODEL: márgenes

En el siguiente ejemplo vamos a ver cómo asignarle márgenes y colores a un box o caja:

HTML

Suponiendo que tenemos dos box model en html:

```
<div class="box box1"></div>

    <div class="box box2">Lorem ipsum dolor sit amet consectetur
adipisicing elit. Optio doloremque enim placeat quae necessitatibus
pariatur nemo nihil asperiores cumque officiis delectus perferendis,
fugiat nostrum error. Officiis perferendis ipsa quod porro!</div>
```

En CSS podemos configurar colores y márgenes de la siguiente forma:

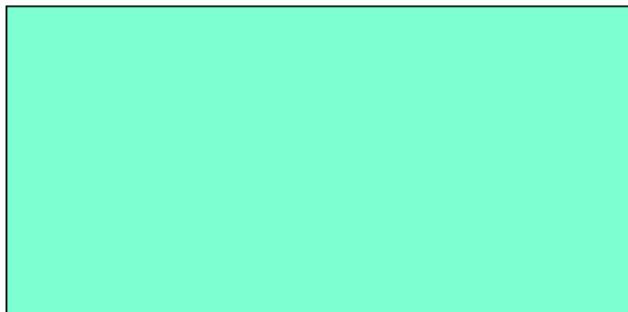
```
.box {
    width: 300px;
    height: 150px;
}

.box1 {
    background-color: aquamarine;
    border: 1px solid black;
    margin-top: 50px;
    margin-bottom: 30px;
    margin-right: 25px;
    margin-left: 30px;
}
```

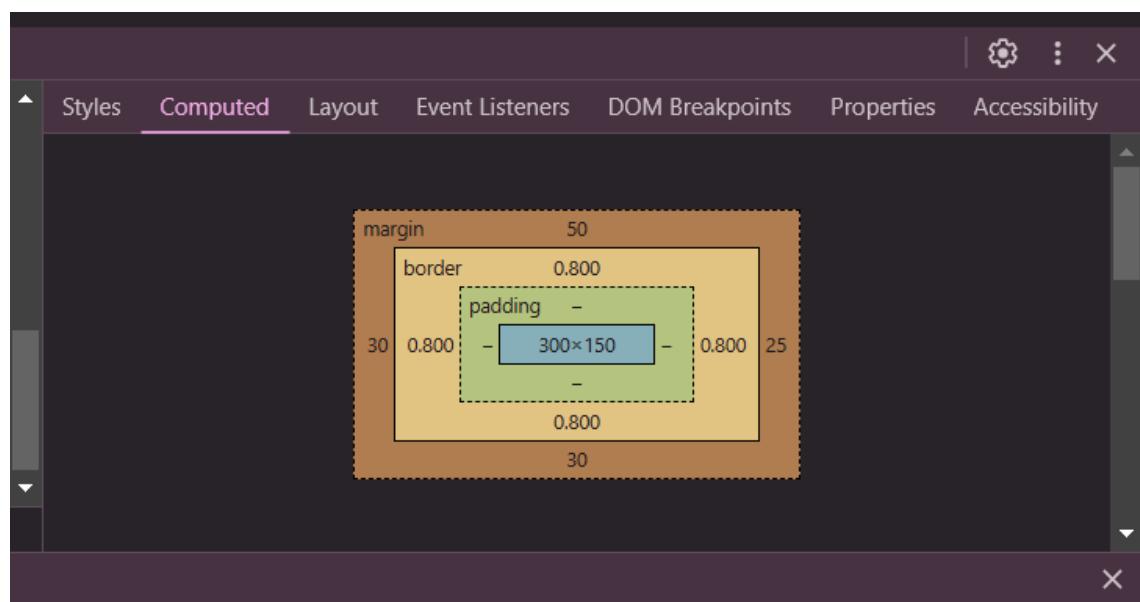
El primer estilo se aplica a ambos box debido a que ambas tienen como class name “box”.

El segundo de los estilos sólo se aplica a box1.

Quedaría de la siguiente forma:



*Lorem ipsum dolor sit amet consectetur
 adipisicing elit. Optio doloremque enim
 placeat quae necessitatibus pariatur nemo nihil
 asperiores cumque officiis delectus
 perferendis, fugiat nostrum error. Officiis
 perferendis ipsa quod porro!*



También podemos asignar márgenes de la siguiente forma:

```
margin: 50px 30px 25px 30px; /*arriba derecha abajo izquierda*/  
  
margin: 40px 80px; /*primer valor para arriba y derecha, segundo valor  
para abajo e izquierda*/  
  
margin: 40px 0 80px 0; /*también así*/
```

```
margin: 40px 50px 80px; /*arriba derecha abajo. Si no se especifica el  
valor de la izquierda, por defecto toma el de la derecha*/  
  
margin: 0 auto; /*con auto lo centra. Para centrar el elemento, debe  
tener un ancho*/
```

Colapso de márgenes

El colapso de márgenes en CSS (margin collapsing) ocurre cuando los márgenes verticales de elementos adyacentes se combinan en lugar de sumarse, resultando en un solo margen igual al valor más grande, no a la suma de ambos.

El colapso de márgenes ocurre sólo en márgenes verticales (margin-top y margin-bottom), no en márgenes horizontales (margin-left y margin-right).

¿Cuándo ocurre el colapso de márgenes?

1. Entre elementos adyacentes (hermanos):

Si dos elementos están uno debajo del otro, sus márgenes verticales se combinan en lugar de sumarse.

html

 Copy  Edit

```
<div class="box1"></div>  
<div class="box2"></div>
```

css

 Copy  Edit

```
.box1 {  
    margin-bottom: 30px;  
}  
.box2 {  
    margin-top: 20px;  
}
```

Resultado: El margen total entre ambos no será $30px + 20px = 50px$, sino solo $30px$ (el valor mayor).

2. Entre padre e hijo (márgenes externos):

Si un elemento hijo tiene un margin-top y no hay padding ni border en el contenedor, ese margen puede "salirse" y afectar al padre.

3. Entre un elemento vacío y su margen:

Si un elemento no tiene contenido, su margen superior e inferior pueden colapsar en uno solo.

Overflow y box-sizing

Las propiedades overflow y box-sizing en CSS controlan el comportamiento del contenido dentro de un contenedor y cómo se calcula su tamaño.

1. overflow

La propiedad overflow define qué sucede si el contenido de un elemento es más grande que el área visible (contenedor).

Valores comunes:

- visible (por defecto): El contenido desborda sin ocultarse.
- hidden: El contenido desbordado se oculta sin mostrar barras de desplazamiento.
- scroll: Muestra barras de desplazamiento siempre, aunque no haya desbordamiento.
- auto: Muestra barras de desplazamiento solo si el contenido desborda.
- clip: Similar a hidden, pero sin posibilidad de desplazamiento (más eficiente)

¿Cuándo usarlo?

hidden: Para ocultar contenido sobrante sin scroll.

auto: Para agregar scroll solo cuando sea necesario.

scroll: Si prefieres barras visibles siempre.

2. box-sizing

La propiedad box-sizing define cómo se calculan las dimensiones totales de un elemento, considerando o no el padding y el border dentro del ancho y alto especificados.

Valores comunes:

- content-box (por defecto): El ancho y alto se calculan sólo para el contenido, excluyendo el padding y el border.
- border-box: El ancho y alto incluyen contenido, padding y border.

¿Cuándo usarlo?

border-box es más práctico para diseños responsivos y evitar desbordes inesperados.

BORDES

<https://developer.mozilla.org/es/docs/Web/CSS/border-style/>

<https://border-radius.com/>

Podemos ver los distintos tipos de bordes que se pueden aplicar en CSS a través de un ejemplo:

```
/*border: width style color;*/

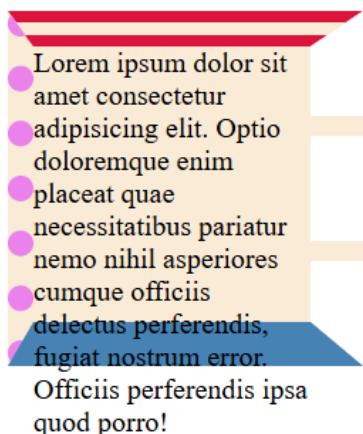
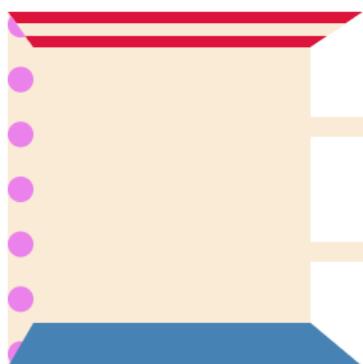
border-top-width: 20px;
border-top-style: double;
border-top-color: crimson;

border-right-width: 30px;
border-right-style: dashed;
border-right-color: white;

border-bottom-width: 25px;
border-bottom-style: solid;
border-bottom-color: steelblue;

border-left-width: 15px;
border-left-style: dotted;
border-left-color: violet;
}
```

Se vería de la siguiente forma:



Bordes aplicados a una imagen

Veamos un ejemplo donde aplicamos distintos tipos de bordes a una imagen:

En CSS

```
.img{  
    display: block;  
    width: 200px;  
    height: 200px;  
    object-fit: cover; /*adapta la imagen para que no se estire*/  
    margin: 20px auto;  
    border-radius: 50%; /*redondea completamente siempre que haya un  
mismo alto y ancho*/
```

```
border-top-left-radius: 50px;  
border-bottom-left-radius: 20px;  
}
```

En la web se vería de la siguiente forma:



[Playa del Carmen](#)

[Entra ahora](#)



DISPLAY Y FLOTAR ELEMENTOS

DISPLAY INLINE

Todos los elementos dentro del HTML por una cuestión natural, por ser un elemento HTML van a tener su propio display. Hay elementos que van a tener display inline por defecto, hay otros que van a tener display inline, block, hay otros que van a tener display block, etc.

Son de tipo inline los enlaces, las imágenes, el strong, los link, los span, etc.



En la imagen tenemos 4 elementos que están contenidos dentro de un contenedor ("display: inline"). Todos los elementos se ven en una misma línea hasta que no entran más dentro del contenedor y deben pasar a la línea inmediatamente inferior.

Si el contenedor fuera lo suficientemente grande, estarían todos los elementos dentro de una misma línea. Como este contenedor no es lo suficientemente grande, tenemos tres elementos dentro de una línea y el cuarto elemento debajo.

DISPLAY BLOCK

Por el contrario, ciertas etiquetas se renderizan en el navegador en líneas independientes, no mezcladas con el resto del texto. Ejemplos de estas etiquetas son los encabezados (`<h1>` hasta `<h6>`), las citas en bloque (`<blockquote>`), por supuesto los párrafos (`<p>`), y quizás la más conocida de todas que es la etiqueta `<div>` usada normalmente para envolver a otros elementos. A estos elementos se les denomina **elementos de bloque**.

```
display: block
```



block

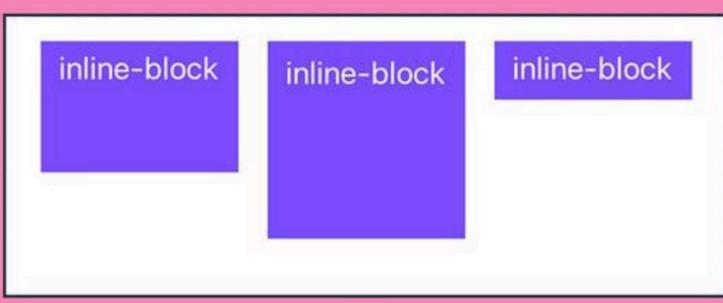
block

Si tenemos dos elementos de tipo block como en la imagen, no pueden ocupar dos espacios en la misma línea, por lo que uno debe ir arriba y el otro debajo. En otras palabras, el elemento ocupa el 100% de la línea en la que se encuentra.

INLINE-BLOCK

Los elementos inline-block fluyen con el texto y demás elementos como si fueran elementos en-línea y además respetan el ancho, el alto y los márgenes verticales.

```
display: inline-block
```



inline-block

inline-block

inline-block



inline

inline



block



inline-block

inline

inline-block



block

Veamos un ejemplo:

Si queremos armar una barra de navegación con ítems que estén uno al lado del otro, deberíamos utilizar el inline. Se vería algo así:



Por más que intentemos colocar un ancho a los elementos para que se vean más organizados y separados, los elementos que son display inline no tienen ancho. El ancho de un elemento display inline está determinado por el ancho de lo que contienen (sea texto o cualquier otro elemento). Por eso en la imagen anterior el ítem 1 es más ancho que el resto de los ítems.

¿Cómo hacemos entonces para que los ítems se vean de forma prolífica y ordenada? A través de un display inline-block que combina propiedades del block con propiedades del inline.

De esta forma, la barra de navegación con los ítems se verían así:



Como el display block sí tiene ancho, podremos ajustar el tamaño a cada ítem de tal forma que se vea ordenado en la barra de navegación.

ETIQUETA DIV (BLOCK)

Un <div> es un elemento de bloque, lo que significa que ocupa todo el ancho disponible de su contenedor. Los elementos de bloque comienzan en una nueva línea y se expanden para llenar el espacio horizontal disponible.

La etiqueta <div> se usa para definir una "división" o un "grupo" en el documento. Es una herramienta esencial para agrupar varios elementos en una sola unidad que se puede manipular con CSS y JavaScript.

Por sí misma, la etiqueta <div> no aplica ningún estilo especial. Los navegadores muestran el contenido dentro de un <div> de manera normal, sin ningún cambio visual a menos que se utilicen estilos CSS.

El `<div>` es útil para agrupar elementos de bloque como encabezados (`<h1>`, `<h2>`, etc.), listas (``, ``, ``), párrafos (`<p>`), tablas (`<table>`), y otros `<div>` o elementos de nivel de bloque.

```
<div>
<p>Ejemplo de cómo funcionaria; la etiqueta div para divisiones.</p>
<p>Puedes insertar el contenido que necesites.</p>
<p>Esta sería la última línea de nuestro texto.</p>
</div>
```

ETIQUETA SPAN (INLINE)

A diferencia de `<div>`, el `` es un elemento en línea, lo que significa que no inicia en una nueva línea y solo ocupa el espacio necesario para su contenido. Los elementos en línea se colocan uno al lado del otro en la misma línea.

La etiqueta `` se usa para aplicar estilos a una parte específica del contenido dentro de otro elemento. No tiene un significado semántico propio y es útil para seleccionar pequeñas porciones de texto o contenido dentro de elementos de bloque o en línea.

Al igual que `<div>`, el `` no aplica ningún estilo especial por defecto. Se utiliza principalmente como un contenedor para aplicar estilos CSS o para manipular partes del contenido con JavaScript.

En resumen, la diferencia entre el `div` y `span` es que `div` se emplea para agrupar elementos HTML en un bloque, en cambio, `span` sirve para agrupar dentro de una línea.

```
<div>Contenido <span> del bloque</span> con elementos en línea</div>
```

PROPIEDAD FLOAT DE CSS

La propiedad CSS `float` ubica un elemento al lado izquierdo o derecho de su contenedor, permitiendo a los elementos de texto y en línea aparecer a su costado. El elemento es removido del normal flujo de la página, aunque aún sigue siendo parte del flujo.

Ejemplo:

float: none; (no hay ningún elemento flotante)

The screenshot shows a browser's developer tools with the CSS panel open. A list of CSS properties is shown on the left, and a preview of the element's layout is on the right. The 'float' property is listed five times, with 'float: none;' being the currently selected item, indicated by a yellow border around its box.

float: none;

float: left;

float: right;

float: inline-start;

float: inline-end;

Float me

As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

float: left; (flotar a la izquierda)

The screenshot shows a browser's developer tools with the CSS panel open. The 'float' property is listed five times, with 'float: left;' being the currently selected item, indicated by a yellow border around its box. The preview on the right shows the text 'Float me' floating to the left of the main paragraph.

float: none;

float: left;

float: right;

float: inline-start;

float: inline-end;

Float me

As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

float: right; (flotar a la derecha)

The screenshot shows a browser's developer tools with the CSS panel open. The 'float' property is listed five times, with 'float: right;' being the currently selected item, indicated by a yellow border around its box. The preview on the right shows the text 'Float me' floating to the right of the main paragraph.

float: none;

float: left;

float: right;

float: inline-start;

float: inline-end;

Float me

As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

Truco para solucionar colapso de contenedores

El clearfix es una técnica en CSS para solucionar problemas de colapso de contenedores cuando se usan elementos flotantes (float).

En HTML

```
<div class="clearfix"></div>
```

En CSS

```
.clearfix {  
    float: none;  
    clear: both;  
}
```

Conclusión:

- Usa clearfix solo si trabajas con float.
- Prefiere Flexbox o Grid para evitar estos problemas.

POSICIONES

TIPOS DE POSICIONES

En CSS, la propiedad position controla cómo se coloca un elemento en la página. Existen cinco valores principales:

1. static (por defecto)

Es la posición predeterminada de todos los elementos.

Se posicionan según el flujo normal del documento.

No admite top, right, bottom ni left.

2. relative (relativa a su posición original)

El elemento se mueve desde su posición original sin afectar a los demás.

Admite top, right, bottom y left.

3. absolute (relativa al primer ancestro relative)

Se sale del flujo normal, es decir, otros elementos no lo ven.

Se posiciona con relación al primer elemento padre con relative, si no hay, se usa <html>.

Admite top, right, bottom, left.

position: absolute elimina la posición original del elemento en el flujo del documento, lo que significa que otros elementos actúan como si no existiera.

4. fixed (fijo en la ventana)

No se mueve al hacer scroll.

Se posiciona respecto al viewport (pantalla del usuario).

Admite top, right, bottom, left.

5. sticky (mezcla de relative y fixed)

Se comporta como relative hasta que se alcanza un límite de scroll, luego se fija como fixed.

Requiere un top, bottom, left o right.

Z-INDEX

El z-index en CSS define la profundidad de los elementos, es decir, qué tan "encima" están unos de otros cuando se superponen. Solo funciona en elementos con position: relative, absolute, fixed o sticky.

Cuanto mayor sea el número, más arriba se coloca el elemento.

Si dos elementos tienen el mismo z-index, el que aparece después en el HTML queda encima.

Los elementos sin z-index tienen un valor implícito de auto, lo que sigue el orden del HTML.

Z-Index negativo: Podés usar valores negativos para enviar un elemento hacia atrás.

Z-Index y contenedores relativos: Si un elemento está dentro de un contenedor con position: relative, su z-index sólo se compara con otros elementos dentro de ese mismo contenedor.

Problemas comunes:

- No funciona en static (posición por defecto).
- Si no hay position: relative | absolute | fixed | sticky, z-index no tendrá efecto.
- El z-index no atraviesa contenedores relative si no tienen un z-index mayor.
- Las propiedades de posicionamiento (position, top, left, z-index, etc.) NO se heredan automáticamente.

Si algo no se superpone como esperabas, verificá:

- Que el elemento tenga position adecuado.
- Si está dentro de un contenedor que lo limite.
- El orden del HTML si los z-index son iguales.

POSITION: FIXED

position: fixed; es un valor de la propiedad position en CSS que permite fijar un elemento en la pantalla, independientemente del scroll.

Características principales de fixed:

- El elemento sale del flujo normal del documento, igual que con absolute.

- Se posiciona respecto al viewport (la ventana visible del navegador), no respecto a su contenedor.
- No se mueve al hacer scroll, permanece fijo en la misma posición.
- Se pueden usar top, right, bottom, y left para definir su ubicación.

Ejemplo:

```
.fija {  
    position: fixed;  
  
    top: 50%; /*es necesario indicarlo porque sino el elemento no  
    aparece en pantalla. Debemos indicarle la posicion que tomara en el  
    acompañamiento del scroll*/  
}
```

PSEUDOCLASES

Las pseudoclases en CSS son palabras clave que se agregan a un selector para definir un estado especial de un elemento sin necesidad de modificar el HTML. Se usan para aplicar estilos en función de la interacción del usuario o de la posición del elemento dentro del DOM.

Ejemplos de pseudoclases comunes:

1. *:hover* – Cuando el usuario pasa el cursor sobre el elemento.

```
button:hover {  
    background-color: lightblue;  
    color: white;  
}
```

Este estilo se aplicará cuando el usuario pase el mouse sobre el botón.

2. *:focus* – Cuando el usuario hace clic en un campo de entrada.

```
input:focus {  
    border: 2px solid blue;  
    outline: none;  
}
```

Cuando el usuario hace clic en el input, el borde cambiará a azul.

3. *:nth-child(n)* – Para seleccionar un hijo específico dentro de un contenedor.

```
p:nth-child(2) {  
    color: red;  
}
```

El segundo <p> dentro de su contenedor se volverá rojo.

4. :first-child y :last-child – Para estilizar el primer o último hijo.

```
li:first-child {  
    font-weight: bold;  
}  
  
li:last-child {  
    color: blue;  
}
```

El primer li tendrá negrita y el último será azul.

5. :checked – Para aplicar estilos a elementos seleccionados en formularios.

```
input[type="checkbox"]:checked {  
    accent-color: green;  
}
```

Cuando un checkbox esté marcado, cambiará de color.

6. :not(selector) – Para excluir elementos.

```
p:not(.especial) {  
    color: gray;  
}
```

Todos los <p> excepto los que tienen la clase .especial serán grises.

7. *:disabled*, *:enabled* y *:required* – Para formularios.

```
input:disabled {  
    background-color: lightgray;  
}  
  
input:required {  
    border: 2px solid red;  
}
```

Los campos disabled estarán en gris y los required tendrán un borde rojo.

8. *:link* – Para estilizar enlaces que aún no han sido visitados.

```
a:link {  
    color: blue;  
    text-decoration: none;  
}
```

Los enlaces no visitados aparecerán en azul y sin subrayado.

9. *:visited* – Para estilizar enlaces que ya fueron visitados por el usuario.

```
a:visited {
```

```
    color: purple;  
}
```

Los enlaces que el usuario ya haya visitado cambiarán a color púrpura.

10. :target – Aplica estilos a un elemento cuando es el objetivo de un enlace con ancla (#id).

HTML

```
<ul>  
  
  <div class="nav_links" id="menu">  
  
    <li>php</li>  
  
    <li>JavaScript</li>  
  
    <li>go</li>  
  
    <li>c</li>  
  
  </div>  
  
</ul>  
  
  
<a href="#menu" class="nav_menu">  
  
    
  
</a>
```

CSS

```
/*evento target*/  
  
.nav_links:target {  
  
  background-color: aqua;  
}
```



PSEUDOELEMENTOS

Los pseudoelementos permiten modificar partes específicas de un elemento sin necesidad de agregar nuevas etiquetas en el HTML. Se usan con ::

1. ::before (Antes del contenido de un elemento)

Crea contenido antes del texto del elemento sin modificar el HTML.

2. ::after (Después del contenido de un elemento)

Crea contenido después del texto del elemento.

3. ::first-letter (Estiliza la primera letra de un texto)

Se usa para diseñar la primera letra de un bloque de texto, útil en estilos de revistas o libros.

::first-line (Estiliza la primera línea de un texto)

Solo aplica a la primera línea de un bloque de texto.

::selection (Estiliza el texto seleccionado por el usuario)

Cuando el usuario selecciona un texto con el mouse, se cambia su apariencia.

Importante:

- ::before y ::after requieren content: " "; para funcionar.
- No se pueden aplicar ::first-letter y ::first-line a elementos inline como .

TRANSICIONES

Las transiciones en CSS permiten animar cambios de propiedades de un elemento de manera suave en lugar de hacerlo instantáneamente. Se utilizan para mejorar la experiencia del usuario en una página web.

1. Propiedad transition

La propiedad transition define cómo se realizarán los cambios en las propiedades de un elemento.

```
selector {  
    transition: propiedad duración función_de_tiempo retardo;  
}
```

Parámetros:

propiedad → Propiedad a animar (ej: background-color, width, opacity).

duración → Tiempo de duración de la animación (ej: 0.5s, 2s).

función_de_tiempo → Cómo se comporta la animación (ease, linear, etc.).

retardo (opcional) → Tiempo antes de que comience la animación.

2. Ejemplos

Cambia el color de fondo cuando el usuario pasa el mouse sobre el botón.

```
#boton{  
    display: block;  
    width: 200px;  
    padding: 15px;  
    border: 1px solid red;  
    background-color: azure;  
    margin: 20px;  
}
```

```
text-align: center;
text-transform: uppercase;
font-family: Impact, Haettenschweiler, 'Arial Narrow Bold',
sans-serif;
color: rgb(255, 0, 0);
transition: background-color, width, 5s;
}

/*Para que funcione una transicion se necesita de una pseudoclase*/

#boton:hover{
background-color: aquamarine;
width: 400px;
}
```

Otro ejemplo:

```
#caja{
background-color: blanchedalmond;
width: 200px;
height: 200px;
clip-path: circle(50% at top left);
margin: 20px;
transition-property: clip-path;
transition-duration: 2s; /*tiempo que dura la transicion*/
transition-timing-function: ease; /*velocidad de la transicion*/
```

```
    transition-delay: 1s; /*tiempo de espera para que la transicion se
ejecute*/
}

/*ahora hay que agregar ua pseudoclase para que la transicion
funcione*/

.container:hover #caja {

    clip-path: circle(150% at top left);
}
```

3. Transición en varias propiedades

Puedes aplicar la animación a múltiples propiedades separándolas con comas.

```
div {

    width: 100px;
    height: 100px;
    background-color: green;
    transition: width 1s, height 1s, background-color 1s;
}

div:hover {

    width: 200px;
    height: 200px;
    background-color: orange;
}
```

Cuando el usuario pasa el cursor, el div se agranda y cambia de color en 1 segundo.

4. transition-timing-function (Funciones de tiempo)

Define la velocidad del cambio durante la transición.

Valor	Descripción
<code>linear</code>	Cambio constante sin aceleración
<code>ease</code>	Cambio suave, rápido al inicio y final lento (<i>por defecto</i>)
<code>ease-in</code>	Comienza lento y acelera
<code>ease-out</code>	Comienza rápido y desacelera
<code>ease-in-out</code>	Comienza y termina lento

Resumen

Propiedad	Función
<code>transition</code>	Define la transición de una propiedad
<code>transition-property</code>	Especifica qué propiedad animar
<code>transition-duration</code>	Duración de la animación
<code>transition-timing-function</code>	Tipo de aceleración o desaceleração
<code>transition-delay</code>	Retardo antes de iniciar la animación

ANIMACIONES

Las animaciones en CSS permiten crear efectos dinámicos sin usar JavaScript. Se definen con `@keyframes`, que describe cómo debe cambiar un elemento a lo largo del tiempo.

1. Sintaxis de `@keyframes`

```
@keyframes nombreAnimacion {  
    from { propiedad: valor_inicial; }  
    to { propiedad: valor_final; }  
}
```

from → Estado inicial.

to → Estado final.

Se puede usar porcentajes (%) para definir múltiples pasos.

2. Aplicando la animación a un elemento

Después de definir `@keyframes`, aplicamos la animación con la propiedad `animation`.

```
div {  
    width: 100px;  
    height: 100px;  
    background-color: blue;  
    animation: cambioColor 2s infinite;  
}  
  
@keyframes cambioColor {  
    from { background-color: blue; }  
    to { background-color: red; }  
}
```

```
}
```

Este código hace que el div cambie de azul a rojo cada 2 segundos de manera infinita.

3. Propiedades de animation

```
animation: nombreAnimacion duración timing-function retardo iteraciones  
dirección modo-relleno;
```

nombreAnimacion → Nombre de la animación definida en @keyframes.

duración → Tiempo que dura la animación (2s, 500ms).

timing-function → Cómo se acelera/desacelera la animación (ease, linear, etc.).

retardo → Tiempo antes de que inicie (1s, 0s).

iteraciones → Cuántas veces se repite (infinite, 3, etc.).

dirección → normal, reverse, alternate, alternate-reverse.

modo-relleno → forwards, backwards, both, none

4. Ejemplo con múltiples pasos (%)

```
@keyframes moverCaja {  
  
    0% { transform: translateX(0); }  
  
    50% { transform: translateX(100px) scale(1.2); background-color: orange; }  
  
    100% { transform: translateX(200px); background-color: red; }  
}  
  
.caja {  
  
    width: 100px;
```

```

height: 100px;
background-color: blue;
animation: moverCaja 3s ease-in-out infinite;
}

```

Explicación:

La caja empieza en x = 0.

A mitad del tiempo (50%), se mueve 100px a la derecha, aumenta de tamaño y cambia a naranja.

Al final (100%), se mueve 200px a la derecha y cambia a rojo.

La animación se repite infinitamente con un efecto suave (ease-in-out).

Resumen:

Valor	Descripción
<code>normal</code>	Se reproduce de inicio a fin (por defecto).
<code>reverse</code>	Se reproduce al revés (de fin a inicio).
<code>alternate</code>	Se reproduce de ida y vuelta (inicio-fin, fin-inicio).
<code>alternate-reverse</code>	Igual que <code>alternate</code> , pero empieza al revés.

Propiedad	Descripción
<code>animation-name</code>	Nombre de la animación (<code>@keyframes</code>)
<code>animation-duration</code>	Duración (<code>2s</code> , <code>500ms</code>)
<code>animation-timing-function</code>	Suavizado (<code>ease</code> , <code>linear</code> , etc.)
<code>animation-delay</code>	Tiempo antes de iniciar (<code>2s</code> , <code>0s</code>)
<code>animation-iteration-count</code>	Número de repeticiones (<code>infinite</code> , <code>1</code> , <code>3</code>)
<code>animation-direction</code>	Sentido (<code>normal</code> , <code>reverse</code> , <code>alternate</code>)
<code>animation-fill-mode</code>	Mantiene estado final (<code>forwards</code> , <code>both</code>)

Ejemplos de animaciones trabajadas en el curso:

```
@keyframes desplazamiento {  
    0%  {  
        margin-left: 0px;  
        transform: rotate(0deg);  
    }  
    50% {  
        margin-left: 1900px;  
        transform: rotate(370deg);  
        border-radius: 999px;  
    }  
    100% {  
        margin-left: 0px;  
    }  
}
```

```
#caja2 {  
    margin-top: 25px;  
    width: 250px;  
    height: 250px;  
    background-color: darkmagenta;  
    color: aliceblue;  
    border: 1px solid black;  
    line-height: 200px;  
    text-align: center;  
    font-size: 20px;  
    /*aplicamos animacion*/
```

```
animation-name: desplazamiento;  
animation-duration: 10s; /*tiempo de la animacion*/  
animation-iteration-count: infinite; /*la animacion se hace infinitamente*/  
animation-timing-function: linear; /*velocidad de la animacion*/  
}  
  
/*para hacer una animacion primero defino los @keyframes*/  
  
@keyframes desplazamiento {  
    from {  
        margin-left: 0px;  
    }  
    to {  
        margin-left: 1200px;  
    }  
}
```

Flexbox en CSS

Flexbox (Flexible Box Layout) es un modelo de diseño en CSS que facilita la alineación y distribución de elementos dentro de un contenedor, incluso cuando su tamaño es dinámico. Se usa principalmente para estructuras de una dimensión (filas o columnas).

1. Características principales

- **Diseño flexible y adaptable:** Permite que los elementos cambien de tamaño automáticamente según el espacio disponible.
- **Distribución sencilla:** Facilita la alineación y justificación de elementos sin necesidad de hacks como float o position.
- **Control sobre el espacio entre elementos:** Gracias a propiedades como gap, justify-content y align-items.
- **Órdenes personalizados:** Se pueden reordenar los elementos sin modificar el HTML con order.
- **Soporte para crecimiento y reducción de elementos:** Con propiedades como flex-grow, flex-shrink y flex-basis.

2. Activar Flexbox en un contenedor

Para usar Flexbox, el elemento padre debe tener display: flex o display: inline-flex.

```
.container {  
    display: flex;  
}
```

Esto convierte a los elementos hijos en flex items y permite aplicar las propiedades de Flexbox.

3. Ejes en Flexbox

Flexbox trabaja con dos ejes principales:

- **Eje principal (main axis):** Determinado por flex-direction.
- **Eje secundario (cross axis):** Perpendicular al principal.

flex-direction	Eje Principal	Eje Secundario
row (por defecto)	Horizontal (izquierda a derecha)	Vertical
row-reverse	Horizontal (derecha a izquierda)	Vertical
column	Vertical (arriba a abajo)	Horizontal
column-reverse	Vertical (abajo a arriba)	Horizontal

Diferencias entre los ejes en Flexbox:

1. Eje principal (main axis):

Está determinado por flex-direction.

Se controla con justify-content.

Ejemplo: si flex-direction: row;, el eje principal es horizontal.

Ejemplo: si flex-direction: column;, el eje principal es vertical.

2. Eje secundario / transversal (cross axis):

Es perpendicular al eje principal.

Se controla con align-items (para todos los elementos) o align-self (para un solo elemento).

Ejemplo: si flex-direction: row;, el eje secundario es vertical.

Ejemplo: si flex-direction: column;, el eje secundario es horizontal.

4. Distribución de elementos en el eje principal (justify-content)

Controla la alineación en el eje principal.

Valor	Descripción
<code>flex-start</code> (default)	Elementos alineados al inicio
<code>flex-end</code>	Elementos alineados al final
<code>center</code>	Elementos centrados
<code>space-between</code>	Espaciado máximo entre elementos
<code>space-around</code>	Espaciado uniforme con margen
<code>space-evenly</code>	Espaciado exactamente igual

5. Alineación en el eje secundario (align-items)

Alinea los elementos en el eje perpendicular.

Valor	Descripción
<code>stretch</code> (default)	Los elementos se expanden al máximo
<code>flex-start</code>	Alineado arriba o a la izquierda
<code>flex-end</code>	Alineado abajo o a la derecha
<code>center</code>	Centrado
<code>baseline</code>	Alinea el texto de los elementos

Ejemplo:

```
.container {
    display: flex;
    justify-content: space-between;
}
```

align-self y el eje secundario:

align-self sobrescribe align-items solo para el elemento al que se le aplica y lo alinea en el eje secundario.

Por ejemplo:

```
#layout {  
    display: flex;  
    flex-direction: row; /* Eje principal: horizontal */  
    align-items: flex-start; /* Alinea los elementos arriba */  
}  
  
.blue {  
    align-self: center; /* Se mueve al centro en el eje secundario  
(vertical) */  
}
```

Si el contenedor #layout tiene suficiente altura, .blue se moverá verticalmente al centro.

Conclusión: align-self controla la posición en el eje secundario, mientras que justify-content controla la posición en el eje principal.

6. Control de tamaño de los elementos (flex-grow, flex-shrink, flex-basis)

- **flex-grow:** Controla cuánto crece un elemento en relación con los demás. Siempre el número más alto es el que más grande será de tamaño. Tiene prioridad el flex-grow más alto y aunque le ponga un número altísimo, siempre va a buscar adaptar el tamaño con las demás cajas. Por lo tanto, puedo poner el número 1000 y seguirá siendo del mismo tamaño o un poco mayor.
- **flex-shrink:** Determina cuánto se encoge un elemento si hay poco espacio.
- **flex-basis:** Especifica el tamaño inicial de un elemento antes de aplicar flex-grow o flex-shrink.

```
.item {  
    flex-grow: 1;  
}
```

7. Orden de los elementos (order)

Cambia el orden visual de los elementos sin modificar el HTML.

Ejemplo de uso:

```
.item1 {  
    order: 2;  
}  
  
.item2 {  
    order: 1;  
}
```

En este caso, item2 se mostrará antes que item1.

8. Alineación de múltiples líneas (align-content)

Si hay varias líneas en el contenedor (flex-wrap: wrap), align-content controla su distribución.

Valor	Descripción
flex-start	Agrupa las líneas arriba
flex-end	Agrupa las líneas abajo
center	Alinea al centro
space-between	Espaciado máximo entre líneas
space-around	Espaciado uniforme

Ejemplo:

```
.container {  
    display: flex;  
    flex-wrap: wrap;  
    align-content: center;
```

}

9. Diferencia entre Flexbox y Grid

Propiedad	Flexbox	Grid
Estructura	Una dimensión (fila o columna)	Dos dimensiones (filas y columnas)
Distribución	Espaciado flexible	Control total del layout
Óptimo para	Alineación y distribución de elementos	Layouts complejos

10. ¿Cuándo usar Flexbox?

- Cuando necesitas alinear elementos en una sola dimensión (fila o columna).
- Para centrar elementos de forma sencilla.
- Para botones en un navbar o listas de elementos flexibles.
- Para formularios y tarjetas con distribución adaptable.

Flexbox es una herramienta poderosa y flexible para crear interfaces responsivas sin necesidad de float o position. Se recomienda para alineaciones en una dimensión, mientras que Grid es más útil para estructuras en dos dimensiones.

FLEX WRAP

flex-wrap pertenece a la categoría de propiedades del contenedor flex porque afecta a cómo los elementos se distribuyen dentro del contenedor cuando no caben en una sola línea.

¿Qué hace flex-wrap?

Controla si los elementos flexibles se quedan en una sola línea o si pueden dividirse en varias líneas cuando el espacio es insuficiente.

Valores de flex-wrap

- **nowrap (por defecto):** Todos los elementos se mantienen en una sola línea, aunque se reduzcan de tamaño.
- **wrap:** Permite que los elementos pasen a una nueva línea si no caben en el contenedor.
- **wrap-reverse:** Igual que wrap, pero la nueva línea aparece arriba en lugar de abajo.

EJEMPLOS DE CLASES

```
/* con esta sola linea me ahorro de poner flex direction: row y
flex wrap: wrap. Puedo poner column tambien */

flex-flow: row wrap;

/* en ambos casos alinea el contenido a la izquierda */

justify-content: left;
justify-content: start;

/* contenido alineado a la derecha */

justify-content: start;

/* contenido alineado al centro */

justify-content: center;

/* da espacios entre cajas*/

justify-content: space-around;

/*da espacios entre cajas juntando un poquito mas*/

justify-content: space-evenly;

/*da espacios entre cajas pero a la primera y a la ultima caja las
pega contra el costado del viewport*/

justify-content: space-between;

/* ubico las cajas en el centro del contenedor*/

align-items: center;
```

```
/* ubico las cajas debajo del contenedor*/
align-items: flex-end;

/* ubico las cajas arriba del contenedor*/
align-items: flex-start;
border: 3px solid black;
padding: 5px;

.blue {
    order: 1;
    background-color: blue;
    align-self: center;
    /* funciona siempre que el contenedor principal tenga align-items y
algun valor */
}
```

CSS Grid Layout

CSS Grid Layout es un sistema de diseño bidimensional que permite organizar elementos en filas y columnas de manera flexible y eficiente. A diferencia de Flexbox, que se enfoca en una sola dirección (fila o columna), Grid permite trabajar en ambas direcciones simultáneamente.

Características Principales de Grid

- **Sistema bidimensional** → Permite distribuir elementos en filas y columnas al mismo tiempo.
- **Facilita la alineación** → Ofrece opciones avanzadas de alineación sin necesidad de usar position.
- **Distribución flexible** → Se pueden definir tamaños de columnas y filas fijos, automáticos o proporcionales.
- **Control del espacio entre elementos** → Usa gap para definir la separación sin margin o padding.
- **Permite áreas nombradas** → Se pueden definir regiones específicas con grid-template-areas.
- **Responsive sin media queries** → Propiedades como auto-fit y auto-fill permiten diseños adaptables.
- **Compatibilidad con Flexbox** → Grid y Flexbox pueden combinarse para lograr diseños más dinámicos.

Propiedades Principales de Grid

Contenedor (display: grid o display: inline-grid):

El contenedor Grid se define con display: grid o display: inline-grid para elementos en línea.

```
.container {  
    display: grid;  
  
    grid-template-columns: 200px 200px 200px; /* 3 columnas de 200px */  
  
    grid-template-rows: 100px 100px; /* 2 filas de 100px */  
  
    gap: 10px; /* Espacio entre elementos */
```

```
}
```

Definiendo Columnas y Fila

grid-template-columns y grid-template-rows:

Define el tamaño de las filas y columnas.

```
.container {  
    display: grid;  
  
    grid-template-columns: 1fr 2fr 1fr; /* Tres columnas con  
proporciones */  
  
    grid-template-rows: 100px auto; /* La segunda fila se ajusta  
automáticamente */  
}
```

- **px** → Tamaño fijo.
- **%** → Porcentaje del contenedor.
- **fr** → Fracción del espacio disponible.
- **auto** → Se ajusta al contenido.

Espaciado Entre Elementos (gap, row-gap, column-gap)

Define el espacio entre filas y columnas.

```
.container {  
    display: grid;  
  
    grid-template-columns: repeat(3, 1fr);  
  
    grid-template-rows: repeat(2, 100px);  
  
    gap: 20px; /* Espacio uniforme entre todos los elementos */  
}
```

Ubicación de Elementos (grid-column, grid-row). Grid Column Start y Grid Column End

Permite posicionar elementos dentro de la grilla.

```
.item1 {  
  
    grid-column: 1 / 3; /* Ocupa desde la columna 1 hasta la 3 */  
  
    grid-row: 1 / 2; /* Ocupa la primera fila */  
}
```

También puedes usar span para indicar cuántas columnas o filas debe ocupar:

```
.item2 {  
  
    grid-column: span 2; /* Ocupa 2 columnas */  
  
    grid-row: span 1; /* Ocupa 1 fila */  
}
```

Por otro lado, las propiedades grid-column-start y grid-column-end permiten definir en qué columna comienza y en cuál termina un elemento dentro del grid.

```
.item {  
  
    grid-column-start: 1; /* Inicia en la columna 1 */  
  
    grid-column-end: 3; /* Termina en la columna 3 (sin incluirla) */  
}
```

Esto es equivalente a:

```
.item {  
  
    grid-column: 1 / 3;  
}
```

grid-column-start: 1; → Comienza en la primera columna.

grid-column-end: 3; → Termina antes de la tercera columna (es decir, ocupa dos columnas).

Áreas Nombradas (grid-template-areas)

Permite definir áreas específicas en la grilla.

```
.container {  
    display: grid;  
    grid-template-columns: 1fr 2fr;  
    grid-template-rows: 100px auto 50px;  
    grid-template-areas:  
        "header header"  
        "sidebar content"  
        "footer footer";  
}  
  
.header { grid-area: header; }  
.sidebar { grid-area: sidebar; }  
.content { grid-area: content; }  
.footer { grid-area: footer; }
```

Esto hace que los elementos ocupen las áreas asignadas sin necesidad de definir grid-column y grid-row.

Alineación de Elementos

Se pueden alinear elementos dentro de Grid con justify-items, align-items y place-items.

```
.container {  
    display: grid;  
    grid-template-columns: repeat(3, 1fr);  
    height: 300px;
```

```
justify-items: center; /* Alinea horizontalmente */  
  
align-items: center; /* Alinea verticalmente */  
}
```

Para alinear el contenido dentro de un solo elemento:

```
.item {  
  
    justify-self: end; /* Alinea a la derecha */  
  
    align-self: start; /* Alinea arriba */  
}
```

Grid vs. Flexbox: ¿Cuándo Usar Cada Uno?

Característica	Grid	Flexbox
Dirección	Bidimensional (filas y columnas)	Unidimensional (fila o columna)
Alineación	Más opciones de alineación	Más simple y flexible
Espaciado	Usa <code>gap</code>	Usa <code>margin</code> y <code>gap</code>
Uso recomendado	Diseño de páginas completas	Componentes individuales

BOOTSTRAP

Veamos un ejemplo de clase con su explicación:

```
CursoDesarrolladorWeb > 12-bootstrap > index.html > html > body > div.container-fluid
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7      <link rel="stylesheet" href="style.css">
8      <link rel="stylesheet" href="bootstrap/css/bootstrap.min.css">
9  </head>
10 <body>
11     <h1>Aprendiendo Bootstrap en el curso de Full Stack Web Developer</h1>
12     <h2>Esta herramienta la utilizaremos para diseñar</h2>
13
14     <div class="container-fluid">
15         <!-- si es fluid ocupa el 100% del ancho de la pantalla -->
16         <!-- el container de por si es una caja centrada -->
17         <!-- 12 columnas es el maximo -->
18         <div class="row">
19             <header class="col-12 bg-primary text-white p-2">
20                 <h1>aprendiendo boostrap en el curso</h1>
21             </header>
22
23             <nav id="menu" class="col-12 bg-secondary text-white">
24                 <!-- con col 12 especifico que se ocupe el 100% del ancho disponible -->
25                 <!-- especifico que voy a tener una fila con 5 columnas de igual tamaño
26                 w-50 un ancho de 50% junto los li -->
27
28                 <ul class="row w-50">
29                     <!-- define una fila dentro del sistema grid. w-50 significa que ocupa la mitad
30                         de la fila -->
31                     <li class="col">Inicio</li>
32                     <li class="col">Plantilla 1</li>
33                     <li class="col">Plantilla 2</li>
34                     <li class="col"> Plantilla 3</li>
35                     <li class="col">Contacto</li>
36                     <!-- al ser 5 elementos, cada uno ocupa el 20% -->
37             </ul>
```

```

CursoDesarrolladorWeb > 12-bootstrap > index.html > html > body > div.container-fluid > div.row > section.col-12
 2   <html lang="en">
10  <body>
14    <div class="container-fluid">
18      <div class="row">
23        <nav id="menu" class="col-12 bg-secondary text-white">
34          <li class="col">Contacto</li>
35          <!-- al ser 5 elementos, cada uno ocupa el 20% -->
36        </ul>
37
38      </nav>
39
40      <section class="col-12">
41
42        <div class="row">
43
44          <section class="col-9 border">
45            <h2>Cajas</h2>
46            <hr>
47            <!-- m-2 es un margen -->
48            <div class="row">
49              <div class="col bg-primary m-2">caja 1</div>
50              <div class="col bg-warning m-2">caja 2</div>
51              <div class="col bg-danger m-2">caja 3</div>
52            </div>
53          </section>
54
55        </div>
56
57        <section class="col-9 border">
58
59          <div class="row">
60            <div class="item col col-lg col-md-12 col-sm-12 bg-primary">item 1</div>
61            <div class="item col col-lg col-sm-3 bg-warning">item 2</div>
62            <div class="item col col-lg col-sm-3 bg-danger">item 3</div>
63
64        </section>
65

```

Puntos importantes sobre Bootstrap y lo que se desarrolla en el código

Uso de Bootstrap para el diseño responsive

- Se utiliza Bootstrap para organizar el contenido en una cuadrícula de 12 columnas.
- Se implementa el sistema de clases de Bootstrap para lograr un diseño adaptativo.

Diferencia entre .container y .container-fluid

- Se usa .container-fluid, lo que hace que el contenido ocupe el 100% del ancho de la pantalla en lugar de estar centrado con márgenes.

Sistema de filas y columnas (row y col-)

- Cada row (fila) debe contener 12 columnas en total para evitar desbordes.
- Se utilizan clases como col-12, col-9, col, col-lg, col-sm-3 para organizar el contenido en diferentes dispositivos.

Encabezado y navegación (header y nav)

- header ocupa toda la fila con col-12 y tiene un fondo azul (bg-primary) con texto blanco (text-white) y padding (p-2).
- El nav también ocupa el ancho completo (col-12) con un fondo gris (bg-secondary) y texto blanco.

Lista de navegación (ul.row.w-50)

- ul es una fila (row), y su ancho es 50% del nav gracias a w-50.
- Sus elementos li usan col, lo que hace que se distribuyan de forma equitativa dentro de ul.

Uso de clases de espaciado (m-2, p-2)

- m-2 se usa para agregar margen a las cajas dentro de una fila.
- p-2 se usa en el header para agregar padding interno.

Uso de bordes (border)

- Se agregan bordes a section para una mejor visualización de las cajas.

Diseño responsivo con col-lg, col-md, col-sm

- col-lg: Se aplica en pantallas grandes.
- col-md-12: Ocupa todo el ancho en pantallas medianas.
- col-sm-3: Hace que ciertos elementos ocupen un 25% (3 de 12) en pantallas pequeñas.

Carga de Bootstrap y jQuery

- Se incluye jQuery (jquery-3.7.1.min.js) porque algunas funcionalidades de Bootstrap lo requieren.
- Se carga Bootstrap desde un archivo local (bootstrap/css/bootstrap.min.css y bootstrap/js/bootstrap.min.js).